The Inverse Taylor Expansion Problem in Linear Logic

Michele Pagani
Laboratoire Preuves, Programmes et Systèmes
Université Paris Diderot – Paris 7
Paris, France
michele.pagani@pps.jussieu.fr

Christine Tasson
Laboratoire Preuves, Programmes et Systèmes
Université Paris Diderot – Paris 7
Paris, France
christine.tasson@pps.jussieu.fr

***Abstract*—Linear Logic is based on the analogy between algebraic linearity (i.e. commutation with sums and with products with scalars) and the computer science linearity (i.e. calling inputs only once). Keeping on this analogy, Ehrhard and Regnier introduced Differential Linear Logic (DiLL) — an extension of Multiplicative Exponential Linear Logic with differential constructions. In this setting, promotion (the logical exponentiation) can be approximated by a sum of promotion-free proofs of DiLL, via Taylor expansion.**

We present a constructive way to revert Taylor expansion. Precisely, we define *merging reduction* — a rewriting system which merges a finite sum of DiLL proofs into a proof with promotion whenever the sum is an approximation of the Taylor expansion of this proof. We prove that this algorithm is sound, complete and can be run in non-deterministic polynomial time.

***Keywords*-Linear Logic, Differential interaction nets, Denotational semantics, Rewriting systems.**

Introduction

In the 80's, Girard [1] introduced linear logic (LL) — a refinement of intuitionistic and classical logics. One particularity of LL is to be equipped with a pair of dual modalities (the *exponentials* ! and ?) which give a logical status to the operations of erasing and copying data. The idea is that linear proofs (i.e. proofs without exponentials) correspond to programs which call their arguments exactly once, whilst exponential proofs call their arguments at will. The study of LL contributed to unveil the logical nature of resource consumption and initiated a foundational comprehension of resource-related runtime properties of programs.

Linear logic makes an extensive use of jargon borrowed from vector spaces and analysis: linear, dual, exponential, etc. Indeed, at the very start of LL, there was the fundamental intuition that programs should be modeled as analytic functions and approximated by polynomials, representing bounded (although possibly non-linear) computations. This idea can be realized if one succeeds in interpreting a type as a collection A of bits of information and a datum of type A as a vector $\vec{a} = \sum_{a \in A} m_a a$, where each scalar m_a "counts" the multiplicity of the bit a in $\vec{a}$ (see [2]).

This work has been supported by the french ANR CHOCO

Interpreting formulæ of LL as vector spaces is not straightforward, because exponentials generate infinite dimensional spaces. For this reason, the vector spaces must be endowed with a topology yielding a suitable notion of converging sum. In [3], [4] the fundamental intuition of LL becomes concrete. In these models, programs that use their arguments exactly once are interpreted as continuous linear functions and programs that can call their arguments infinitely often are analytic functions. Moreover, analytic functions can be approximated by polynomials through *Taylor expansion* [5]. This approach is enabled by the presence of a derivative operator. A natural question then arises: what is the meaning of such a derivative from the logical viewpoint? Ehrhard and Regnier answer to this question by introducing the *differential linear logic* (DiLL, [6]), and its functional fragment: the *differential λ-calculus* [7].

In LL, only the promotion rule introduces the ! modality. Operationally, the promotion creates inputs that can be called an unbounded number of times. In DiLL three more rules handle the ! modality (*codereliction*, *cocontraction* and *coweakening*) that are the duals of the LL rules dealing with the ? modality (*dereliction*, *contraction* and *weakening*). In particular, codereliction expresses in the syntax the semantical derivative: it releases inputs of type $!A$ that must be called exactly once, so that executing a program f on a "coderelicted" input x amounts to calculate the best linear approximation of f on x. Notice that this imposes non-deterministic choices — if f is made of several subroutines each of them demanding for a copy of x, then there are different executions of f on x, depending on which subroutine is fed with the unique available copy of x. Thus we have a formal sum, where each term represents a possibility. This sum has a canonical mathematical interpretation — it corresponds to the sum obtained by computing the derivative of a non-linear function.

As expected, the Taylor expansion can be imported in the syntactic realm by iterating differentiation [8]. A proof of LL can be decomposed into a formal sum of promotion-free proofs of DiLL. The principle is to decompose a program into a sum of purely "differential programs", all of them containing only bounded (although possibly non-linear) calls to inputs. Understanding the relation between a program

and its Taylor expansion might be the starting point of renewing the logical approach to the quantitative analysis of computation started with the inception of LL.

A program and its Taylor expansion are say equivalent, for at each argument, their evaluations give the same result. Since the information contained in the Taylor expansion is quite redundant, we conjecture that we can reconstruct the original LL net with only finitely many terms appearing in the Taylor expansion. That is why we propose an algorithm that build all the LL nets whose Taylor expansions contain a given finite DiLL sum. However, there are DiLL proofs that do not appear in the Taylor expansion of the same LL proof, they are not *coherent*. One may think of the terms of a DiLL sum as parallel threads of a computation, the sum is coherent whenever these threads can be joined up into a sequential computation, represented by an LL proof. Our algorithm takes a finite sum $\sum_i \alpha_i$ of DiLL proofs as inputs, runs a rewriting reduction, namely the *merging reduction* and returns an LL proof π or falls in a deadlock. We prove that this algorithm is complete (Th. 1) and sound (Th. 2): π is reached if, and only if, $\sum_i \alpha_i$ is in the Taylor expansion of π. The algorithm is non deterministic and can be run in non-deterministic polynomial time (Cor. 4).

In the sum $\sum_i \alpha_i$, each term may occurs several times, i.e. $\sum_i \alpha_i$ can be expressed as a linear combination of DiLL proofs with scalars in $\mathbb{N}$. Although the scalars are needed to get the equivalence between an LL proof and its Taylor expansion, we conjecture that they only depend on the DiLL proofs as it is the case for λ-terms [8]. In particular, with respect to the results achieved in this paper, scalars play no role. Hence we do not tackle coefficients issue, and we will define Taylor expansions as sets of DiLL proofs.

The syntax of nets. We represent LL proofs as graphs called *ll-nets* (Def. 1). In [1], ll-nets are called *proof structures*. The distinction between proof structures and proof nets (that are logically correct proof structures) plays no role in this paper: we will thus omit to speak of any correctness criterion. Besides, we consider only cut-free ll-nets. We adopt the syntax of [9] with generalized contractions and atomic axioms. In addition we have coweakenings, needed to define the informative order (Def. 6) and to state completeness (Th.1). Concerning DiLL, we represent its proofs as *polynets*, which are sets of *simple nets* (Def.2).

Outline. Section I defines the Taylor expansion of ll-nets into polynets (Def.5). In Section II, we define *labelings* (Def.7), an equivalent but more local way to deal with boxes. We present our rewriting system, the terms, called *merging triples* (Def.14), and the reduction over them, called *merging reduction* (Def.13). We prove that the merging reduction is non-deterministically polynomial (Cor. 4), complete (Th. 1) and sound (Th. 2) with respect to the Taylor expansion.

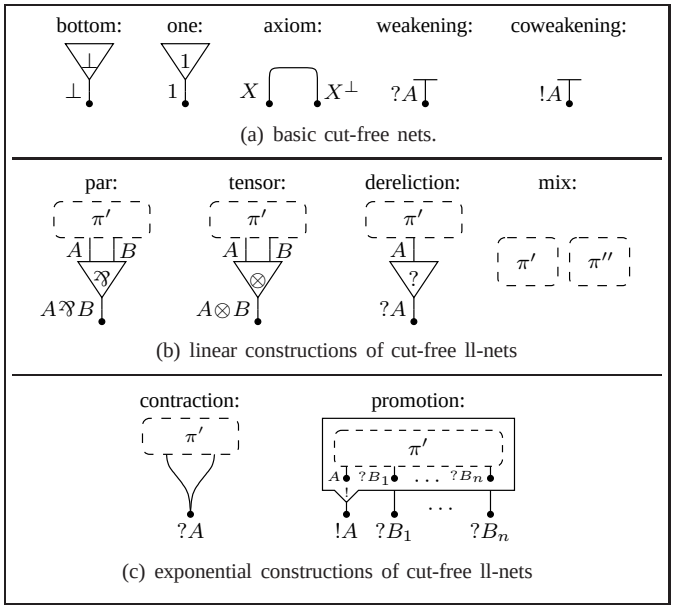


Figure 1: inductive definition of cut-free ll-nets.

I. Taylor expansion: from ll-nets to polynets

We consider formulæ of propositional multiplicative exponential linear logic (MELL), generated by the grammar:

$$A, B := X \mid X^{\perp} \mid 1 \mid A \otimes B \mid \perp \mid A \parr B \mid !A \mid ?A\,,$$

where X, $X^{\perp}$ range over an enumerable set of propositional variables. The linear negation is involutive, i.e. $A^{\perp\perp} = A$, and defined through De Morgan laws $1^{\perp} = \perp$, $(A \otimes B)^{\perp} = A^{\perp} \parr B^{\perp}$ and $(!A)^{\perp} = ?A^{\perp}$.

Definition 1. The cut-free linear logic nets, **ll-nets**[1] for short, are inductively defined by the constructions drawn on Figures 1(a), 1(b) and 1(c), supposing that π' and π'' are cut-free ll-nets. They are finite hypergraphs made of (i) nodes labeled by MELL formulæ and called **ports**; (ii) directed hyperedges labeled by MELL connectives, depicted as triangles and named **cells**; (iii) directed hyperedges crossing ports labeled by a same exponential formula and named **structural wires**; (iv) undirected edges called **simple wires**, they cross two ports labeled by the same formula or (only in the axiom case) labeled by dual formulæ.

A cell/structural wire c has a unique target, named the **principal port** of c, the sources, if any, are called the **auxiliary ports** of c. We adopt the convention of depicting the directed hyperedges with a top-down orientation.

A port of an ll-net π is **free** whenever it is not crossed by any cell. We require that π is given together with an **interface** $(p_i : A_i)_{i \leq n}$ enumerating its free ports with their types. The interfaces $(p_i : A_i)_{i \leq n}$ and $(q_i : B_i)_{i \leq m}$ are **paired** whenever $n = m$ and $A_i = B_i$.

[1] This definition is kept informal. We refer to [6], [10] for precisions.

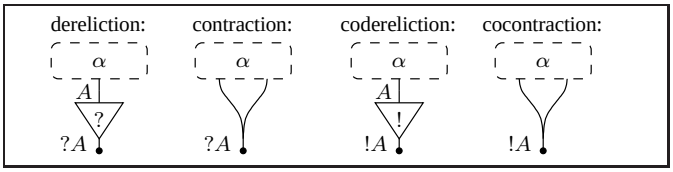
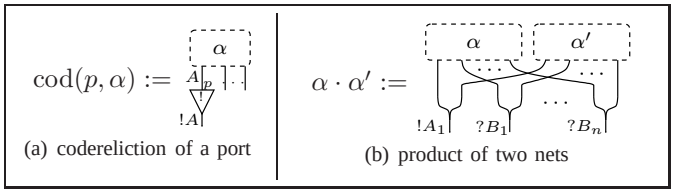


Figure 2: exponential constructions of cut-free simple nets.



(a) codereliction of a port

(b) product of two nets

Figure 3: codereliction and product of simple nets.

(Co)weakenings (Fig. 1(a)) are unary structural wires. In the contraction case (Fig. 1(c)), π' is a cut-free ll-net with at least two free ports p, q of type $?A$; to obtain the drawn ll-net, we equal p, q with a unique free port $r : ?A$ and merge the two (hyper)edges sharing r. In the promotion case, the ll-net π' is put into a **box**; this box is a cell labeled by a cut-free ll-net: its **contents**. Notice that given the box interface $(p_0 : !A, q_1 : ?B_1, \ldots, q_n : ?B_n)$, the interface of its contents is $(p'_0 : A, q'_1 : ?B_1, \ldots, q'_n : ?B_n)$ where the principal ports p_0 and p'_0 and the auxiliary ports q_i and q'_i match. We require moreover that:

$(*)$ any free port $q'_i : ?B_i$ of the contents of a box does not belong to a structural wire.[2]

A cell/wire c is of **depth** 0 in an ll-net π, if c is a (hyper)edge of π view as a hypergraph; c is of depth $d+1$ in π, if there is a box b of depth 0 in π and c is of depth d in the contents of b. The depth of π is the maximal depth of the cell/wires in π. The set of boxes of depth 0 of π is denoted by $\mathrm{box}_0(\pi)$ and the set of boxes at any depth by $\mathrm{box}(\pi)$. We define similarly the set $\mathrm{cell}(\pi)$ of cells at any depth of π. Finally, we denote $c \in \pi$ if $c \in \mathrm{cell}(\pi)$.

Let $b, b' \in \mathrm{box}(\pi)$ and $\pi_b, \pi_{b'}$ be the contents of resp. b and b'. Remark that $\mathrm{cell}(\pi_b)$ and $\mathrm{cell}(\pi_{b'})$ are disjoint or one is included in the other. This means that $\supseteq$ is a **tree-order** over $\{\mathrm{cell}(\pi_b)\ ;\ b \in \mathrm{box}(\pi)\}$, i.e. whenever $\mathrm{cell}(\pi_b)$ and $\mathrm{cell}(\pi_{b'})$ have a sup, then they are comparable.

As mentioned in the Introduction, boxes represent data that can be called infinitely often during the execution of a program. In DiLL new rules (cocontraction and codereliction) deal with !-formulæ but keep bounded the number of calls to the data. This allows to represent non-linear programs as *simple nets* where boxes are replaced by (co)contractions which explicitly give the number of calls to their contents.

Definition 2. The cut-free **simple nets** are inductively defined by the constructions depicted on Fig. 1(a) and 1(b) and by the exponential constructions of Fig. 2. The cocontraction case is defined analogously to the contraction case. A **polynet** is a finite set of simple nets with paired interfaces.

Except for boxes and depth which have no meaning in the context of simple nets, we use the vocabulary of ll-nets. The word **net** will refer equally to ll-nets or simple nets.

[2]This condition is needed to have a canonical representation of ll-nets. It can be equivalently stated as: every q'_i is connected by a simple wire to a dereliction or to an auxiliary port of another box.

The Taylor expansion decomposes an ll-net π into a set $\mathcal{T}(\pi)$ of simple nets; each simple net in $\mathcal{T}(\pi)$ represents an "instance" of π where every box has been replaced by a finite number of copies of its contents. Before giving the definition of $\mathcal{T}(\pi)$ (Def. 5), we introduce substitution (Def. 3), $\mathrm{cod}(p, \alpha)$ and product (Def. 4, Fig. 3).

Definition 3. Let α, β and γ be three nets (possibly with non-atomic axioms) such that β and γ have paired interfaces $(p_i : A_i)$ and $(q_i : A_i)$. If β is a **subnet** of α, denoted $\beta \subseteq \alpha$, then the **substitution** $\alpha[\gamma/\beta]$ is the net obtained from α by replacing β with γ. So, q_i replaces p_i and the wires sharing q_i are merged.[3]

Definition 4. Let p be a free port of a simple net α. We denote as $\mathrm{cod}(p, \alpha)$ the simple net obtained from α by adding a codereliction with auxiliary port p (Fig. 3(a)).

Let α and α' be two simple nets with paired interfaces resp. $(p : !A_1, q_1 : ?B_1, \ldots, q_n : ?B_n)$ and $(p' : !A_1, q'_1 : ?B_1, \ldots, q'_n : ?B_n)$. The **product** $\alpha \cdot \alpha'$ is the simple net resulting from the cocontraction of p and p' and the contractions of q_i and q'_i (Fig. 3(b)).

The product of simple nets is commutative, associative and its neutral element is the net only made of (co)weakenings, that we denote $!\mathbf{0}$.

Definition 5. The **Taylor expansion**[4] of an ll-net π is the set of simple nets $\mathcal{T}(\pi)$ defined by induction on the depth of π (Fig. 4(b)). We distinguish two cases according to whether π is a box b, or a generic ll-net:

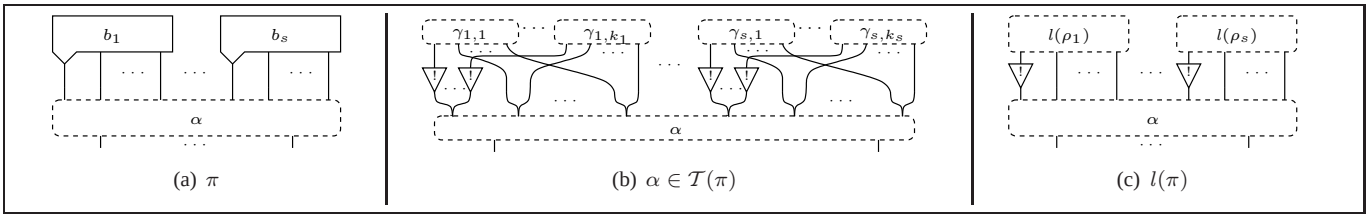
$$\mathcal{T}(b) := \left\{ \prod_{j=1}^{k} \mathrm{cod}(p_j, \gamma_j)\ ;\ \begin{array}{l}\text{with } k \in \mathbb{N},\ \gamma_j \in \\ \mathcal{T}(\rho),\ \rho \text{ the contents of} \\ b \text{ and } p_j \text{ the free port of} \\ \gamma_j \text{ corresponding to the} \\ \text{principal port of } b.\end{array} \right\},$$

$$\mathcal{T}(\pi) := \left\{ \pi[\beta_r/b_r]_{r \le s}\ ;\ \begin{array}{l}\text{with } \mathrm{box}_0(\pi) = \{b_r\}_{r \le s}, \\ \text{and } \beta_r \in \mathcal{T}(b_r)\end{array} \right\}.$$

With the notations of the above definition, notice $!0 \in \mathcal{T}(b)$, since $k = 0$ yields the empty product.

[3]Although intuitively clear, the operation of merging wires should be handled with care because loops and cuts can be produced. We refer to [10] for a formal definition. Indeed, throughout this paper we will deal only with substitutions yielding cut-free and loop-free nets.

[4]Notice that the Taylor expansion defined by Ehrhard and Regnier [8] was given in terms of sums. As written in the Introduction, we will deal only with the supports of these sums.

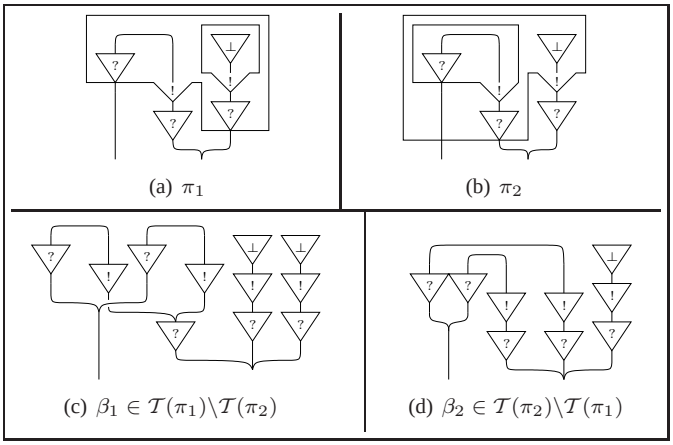


(a) π (b) $\alpha \in \mathcal{T}(\pi)$ (c) $l(\pi)$

Figure 4: (a) an ll-net π s.t. $\mathrm{box}_0(\pi) = \{b_r\}_{r\leq s}$; (b) the generic shape of a simple net $\alpha \in \mathcal{T}(\pi)$; (c) the linearization $l(\pi)$.

Not every polynet is the Taylor expansion of an ll-net. Indeed, simple nets appearing in the Taylor expansion of an ll-net π are "coherent": their structure reflects the boxes of π. In Figures 5(c) and 5(d), we present an example of two incoherent simple nets $\beta_i \in \mathcal{T}(\pi_i)$, $i = 1, 2$. However, π_1 and π_2 have the same linearization which is intuitively obtained by forgetting the contour line of boxes. More formally, the **linearization** $l(\pi)$ (Fig. 4(c)) of an ll-net π is inductively defined by $l(\pi) := \pi[\mathrm{cod}(p_r, l(\rho_r))/b_r]_{r\leq s}$, where $\mathrm{box}(\pi) = \{b_r\}_{r\leq s}$, ρ_r is the contents of b_r and p_r is the principal free port of $l(\rho_r)$.

In the sequel $l(\pi)$ will play an important role, since it describes the structure of π except from the boxes contour line. Indeed, it is a simple net of $\mathcal{T}(\pi)$, obtained by taking exactly one copy of every box of π.



(a) π_1 (b) π_2 (c) $\beta_1 \in \mathcal{T}(\pi_1)\backslash\mathcal{T}(\pi_2)$ (d) $\beta_2 \in \mathcal{T}(\pi_2)\backslash\mathcal{T}(\pi_1)$

Figure 5: π_1 and π_2 with same linearization.

Since the empty product !0 is an ll-net, we can introduce an informative order on ll-nets, useful in the sequel:

Definition 6. We say that an ll-net π' is **less informative** than an ll-net π and we write $\pi' \ll \pi$, whenever there are boxes $(b_r)_{r\leq s}$ in $\mathrm{box}(\pi)$ such that $\pi' = \pi[!0/b_r]_{r\leq s}$.

It is easy to check that $\ll$ is an order. Intuitively, π' is the result of erasing some subroutines of π. In general a finite subset of $\mathcal{T}(\pi)$ does not have enough information to build π and we will rather build $\pi' \ll \pi$ (see Th. 1). However,

$$\pi' \ll \pi \Rightarrow \mathcal{T}(\pi') \subseteq \mathcal{T}(\pi). \qquad (1)$$

II. Reversing Taylor expansion

In this section we present the merging reduction: our algorithm reversing the Taylor expansion. Given a finite polynet, the *initial state* (Def. 12, Fig. 8(b)) is obtained by plugging the simple nets into *counters* (Def. 10, Fig. 8(a)). Then these counters explore the simple nets, merge equal cells and draw boxes when it is possible. If the algorithm succeeds, then the result is an ll-net. On Fig. 9, we give the elementary reduction steps (**ers**) of the merging reduction.

A. An example.

Before going into more details, let us run our algorithm on an example. The rewriting is depicted step by step on Fig. 6. We draw in boldface the redex which is about to be reduced. The run we follow is successful and its result is the ll-net depicted on Fig. 5(a).

Initial state. Consider the polynet $\{\alpha_1, \alpha_2\}$, where $(p_i : ?1, q_i : ?!\bot)$ is the interface of α_i. The algorithm starts from the initial state depicted on Fig. 6(a). Two counters connect α_1 and α_2, one for $?1$ and one for $?!\bot$. There are two *tokens* $\mathtt{l}_1$ and $\mathtt{l}_2$ inside the counters and an *alphabet* $\{\mathtt{A}\}$ containing an *address* $\mathtt{A} = \{\mathtt{l}_1, \mathtt{l}_2\}$ which is the set of tokens inside the counters (Def. 9).

First step. The only possibility is to apply a step `contr` to the right counter, setting $n_1 = 2, n_2 = 3$ and so $m = 2$ (see Fig. 9 for the notation). Indeed we need to choose how to distribute the three auxiliary ports of the contraction of α_2. It is a non-deterministic step of the merging algorithm: different choices may lead to non-confluent reductions. In this example, apart from the reduction we will pursue, one choice *leads to* the ll-net of Fig. 5(b), and the other ones *fail*, i.e. lead to nets with counters that are not further reducible.

Step 2. The derelictions of the redex are merged into a unique dereliction labelled with the address $\mathtt{A}$ (recall that it is the set of the tokens $\mathtt{l}_1, \mathtt{l}_2$ in the merging counter).

Step 3. The next redex is reduced by the "crucial" ers $\xrightarrow{!_\mathtt{p}}$. This step has "created" a box by adding three new tokens $\mathtt{l}_1^1, \mathtt{l}_2^1, \mathtt{l}_2^2$ and a new address $\mathtt{B} = \{\mathtt{l}_1^1, \mathtt{l}_2^1, \mathtt{l}_2^2\}$. The new tokens are associated with the coderelictions in the redex and they extend the old ones in a sense made precise in Def. 9: specifically $\mathtt{l}_1^1$ (resp. $\mathtt{l}_2^1, \mathtt{l}_2^2$) extends $\mathtt{l}_1$ (resp. $\mathtt{l}_2$). The address $\mathtt{B}$ represents a box associated with the codereliction labeled by $\mathtt{B}$ and resulting from the merging of the three coderelictions. The new address opens the possibility of

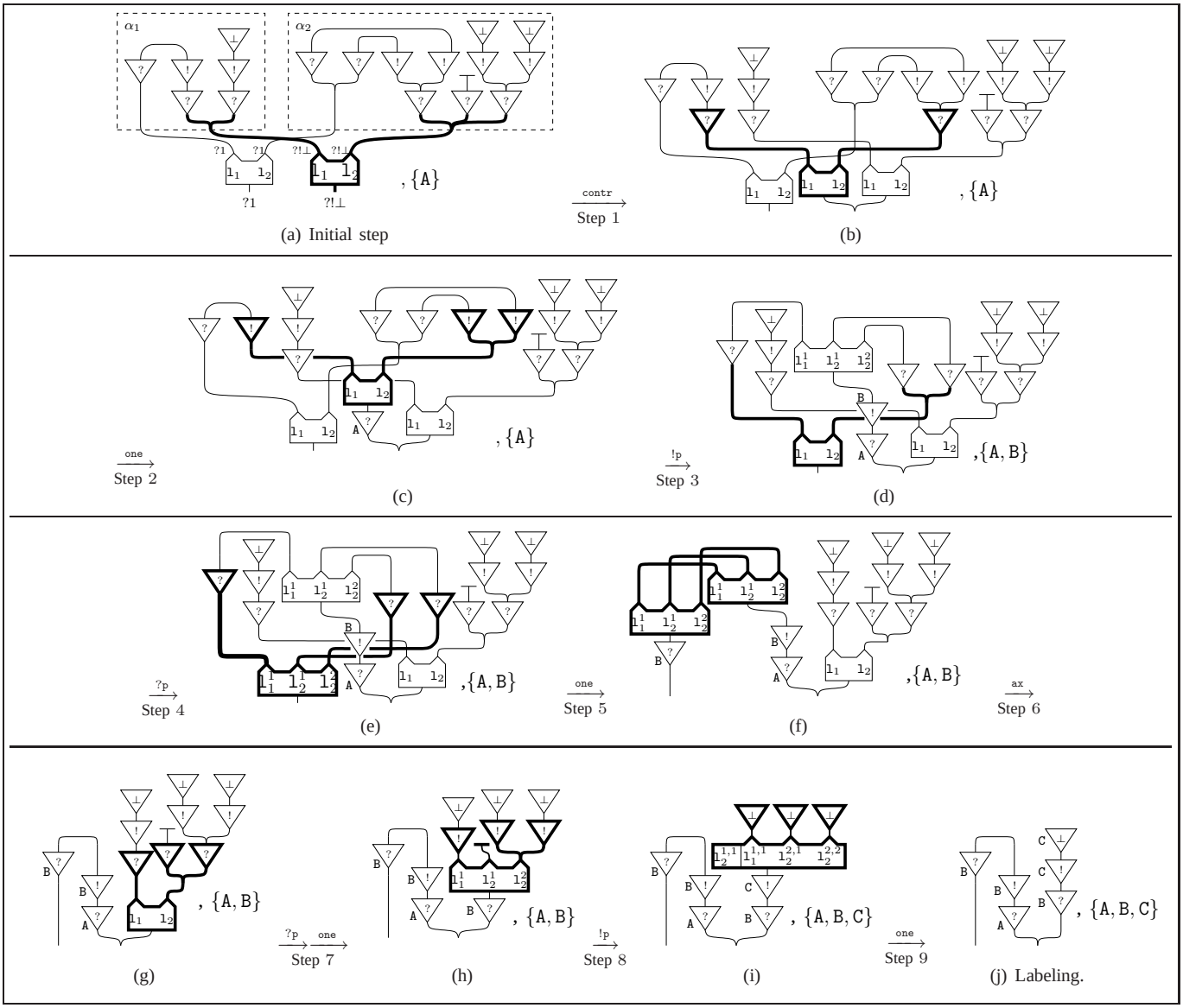


Figure 6: an example of reduction.

applying the ers $\xrightarrow{?p}$ to the two counters inactive until now.

Step 4**.** While $\xrightarrow{!p}$ creates a box adding a new address, and enters it via the principal port, a $\xrightarrow{?p}$ enters a box already created using an address available in the alphabet (here B) and via an auxiliary port. Notice also that a $\xrightarrow{?p}$ can "consume" contractions (here, the counter increases the number of its auxiliary ports) but it does not merge derelictions since they can belong to other boxes.

Step 5**.** The address stored in a counter after a number of $\xrightarrow{?p}$ ers must be put down on a cell by a $\xrightarrow{one}$.

Step 6**.** Two counters meet and they share exactly the same address. Thus they can be eliminated by a $\xrightarrow{ax}$ step.

Step 7**.** The $\xrightarrow{?p}$ enters in the box B, consuming contraction. The ers $\xrightarrow{one}$ merges the derelictions into one dereliction labeled with B.

Step 8**.** Remark that one port of the counter is wired to a coweakening. The $\xrightarrow{!p}$ creates four more tokens $1_1^{1,1}, 1_2^{1,1}, 1_2^{2,1}, 1_2^{2,2}$ associated with the coweakening/coderelictions of the redex. A new address C which is the set of new tokens appears. These tokens extend the old ones as hinted by the indices. The token $1_2^{1,1}$, associated with the coweakening in the redex, is stored in a special basket that will be kept until the counter is erased (Def. 10).

Last step. The resulting net is a *labeling* (Def. 7). It has neither counter nor cocontraction and every cell is labelled by an address. It represents the ll-net drawn in Fig. 5(a).

In order to be as local as possible, our reduction cannot use boxes as they require to define their "frames" all in one

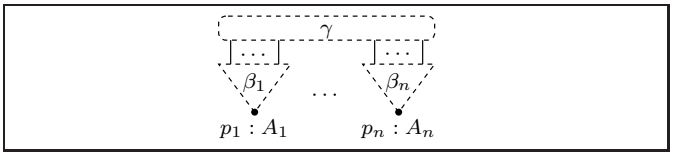


Figure 7: decomposition of a simple net.

go. Thus, we reconstruct the linearization $l(\pi)$ of an ll-net π and we represent the boxes by labeling the cells of $l(\pi)$ with addresses. A total labeling encodes exactly the boxes of π (Prop. 1). During the execution of the merging algorithm, the partial labeling is extended step by step up to a total function. The frames of the boxes of π are recovered from the addresses labeling the cells of $l(\pi)$.

B. Labeling

In our example the box associated with the codereliction labeled by $\mathtt{B}$ contains the cells labeled by $\mathtt{B}$ and every cell above. Notice that the set of addresses is endowed with an order: $A \sqsubseteq B \sqsubseteq C$, which means that the box B contains the box C. Not every labeling is a correct boxing, we give conditions (Def. 7) on labelings sufficient to ensure the equivalence with ll-nets (Prop. 1).

Any cut-free simple net α has a **canonical decomposition** into a subnet γ made of axioms and pairwise disjoint trees $(\beta_i)_{i\leq n}$ made of cells and wires (Fig. 7). The leaves of β_i can be units ($\bot$ or 1), (co)weakenings, or axioms in γ. We set $a \leq_\alpha b$ whenever a, b belong to the same tree and a is an ancestor of b. If α has more than one conclusion then there are several minimals with respect to $\leq_\alpha$. We introduce a **conclusion cell** $\bot_\alpha$ set to be the minimum of $\leq_\alpha$.

Definition 7. Let $\mathcal{N}$ be an infinite set of **names**, let α be a cut-free simple net without cocontraction. We denote $\mathrm{coder}(\alpha)$ the set of codereliction cells of α. Let $\mathfrak{L} : \{\bot_\alpha\} \cup \mathrm{cell}(\alpha) \to \mathcal{N}$ be a total function s.t.:

- $\mathfrak{L}$ is injective on $\mathrm{coder}(\alpha) \cup \{\bot_\alpha\}$;
- the codomain $\mathfrak{L}(\alpha)$ of $\mathfrak{L}$ is $\mathfrak{L}(\mathrm{coder}(\alpha) \cup \{\bot_\alpha\})$.

Let us define $\sqsubseteq^\circ_{\alpha,\mathfrak{L}}$ as the $\mathfrak{L}$ image of $\leq_\alpha$ onto $\mathfrak{L}(\alpha)$, that is:

$$\forall n, m \in \mathfrak{L}(\alpha),\ n \sqsubseteq^\circ_{\alpha,\mathfrak{L}} m \iff \exists\, c \leq_\alpha d,\ \begin{cases} n = \mathfrak{L}(c) \\ m = \mathfrak{L}(d). \end{cases}$$

Let us denote $\sqsubseteq_{\alpha,\mathfrak{L}}$ the transitive closure of $\sqsubseteq^\circ_{\alpha,\mathfrak{L}}$. The pair $(\alpha, \mathfrak{L})$ is called a **labeling** whenever

(i) $\sqsubseteq_{\alpha,\mathfrak{L}}$ is a tree-order with minimum $\mathfrak{L}(\bot_\alpha)$;
(ii) if c has a predecessor c', then either $\mathfrak{L}(c') = \mathfrak{L}(c)$ and c is not a codereliction, or $\mathfrak{L}(c)$ is the son of $\mathfrak{L}(c')$ and c is a codereliction, or finally c is a dereliction;
(iii) given two ports p, q connected by an axiom, if p is an auxiliary port of a cell c s.t. $\mathfrak{L}(c) \neq \mathfrak{L}(\bot_\alpha)$, then q is the auxiliary port of a cell c' s.t. $\mathfrak{L}(c') = \mathfrak{L}(c)$.

From the order induced by the labeling, one can recover the contents of the box associated with a codereliction. Then a labeling and a box match if the contents of their boxes coincide.

Definition 8. Let $(\alpha, \mathfrak{L})$ be a labeling. With any codereliction b of α, we associate the labeling $\mathrm{cont}(\alpha, \mathfrak{L}, b)$ corresponding to its contents. It is defined by the simple net

$$\{c \in \alpha \,|\, \mathfrak{L}(b) \sqsubseteq_{\alpha,\mathfrak{L}} \mathfrak{L}(c),\ c \neq b\},$$

and the labeling $\mathfrak{L}_{\text{cont}}(\bot) = \mathfrak{L}(b)$ and $\mathfrak{L}_{\text{cont}}(c) = \mathfrak{L}(c)$.

We say that a labeling $(\alpha, \mathfrak{L})$ is **equivalent** to an ll-net π and we write $(\alpha, \mathfrak{L}) \equiv \pi$ for short, whenever $\alpha = l(\pi)$ and

$$\forall b \in \mathrm{box}(\pi) \text{ with contents } \rho,\ \mathrm{cont}(\alpha, \mathfrak{L}, b) \equiv \rho. \qquad (2)$$

Proposition 1. *A labeling is equivalent to a unique cut-free ll-net and vice versa (up to a renaming).*

Proof: For any labeling one proves that there is a unique equivalent ll-net by induction on the size of the simple net of the labeling. The converse is proven by building a labeling candidate on the linearization: the order $\sqsubseteq$ reflects the tree-order of boxes, the labeling properties (Def 7) follow. ■

Recall the labeling of Fig. 6. The set of names is $\mathcal{N} = \{\mathtt{A}, \mathtt{B}, \mathtt{C}\}$ and we have $A \sqsubseteq B \sqsubseteq C$, which encodes the nesting of the boxes of the ll-net of Fig. 5(a).

C. Reduction

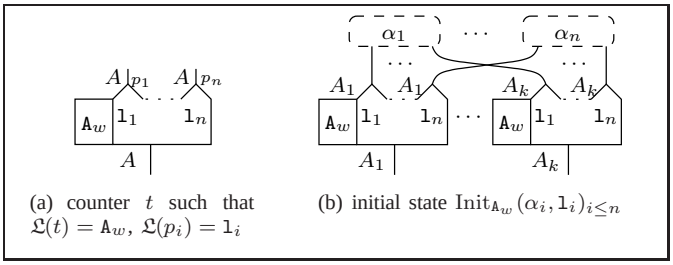
The most delicate task of merging reduction is to reconstruct a correct nesting of boxing, i.e. the order $\sqsubseteq$ of Def.7. This reconstruction is made step by step, using the set theoretical inclusion of tokens, and the induced order $\trianglelefteq$ on addresses (Def.9): at the end of the process we will have $\trianglelefteq = \sqsubseteq$ and consequently an ll-net.

Definition 9. Let X be an enumerable set called the **web**. A **token** is a finite set of elements in X. An **address** is a finite set of tokens. We set $\mathtt{l}, \mathtt{m}$ to range over tokens, $\mathtt{A}, \mathtt{B}$ to range over addresses, and $\mathcal{A}, \mathcal{B}$ to range over sets of addresses. The set-theoretical inclusion of tokens induces the Smyth pre-order on addresses:

$$\mathtt{A} \trianglelefteq \mathtt{B} \iff \forall \mathtt{m} \in \mathtt{B},\ \exists \mathtt{l} \in \mathtt{A},\ \mathtt{l} \subseteq \mathtt{m}.$$

It is immediate to prove that $\trianglelefteq$ is a pre-order. However let us stress that $\trianglelefteq$ is not antisymmetric (consider $\mathtt{A} = \{\mathtt{l}, \mathtt{m}\}$, $\mathtt{B} = \{\mathtt{l}, \mathtt{m}'\}$, with $\mathtt{l} \subset \mathtt{m}, \mathtt{m}'$), nor tree-like (consider $\mathtt{A} = \{\mathtt{l}\}$, $\mathtt{A}' = \{\mathtt{m}\}$, $\mathtt{B} = \{\mathtt{l}, \mathtt{m}\}$, with $\mathtt{l}$, $\mathtt{m}$ disjoint) on the whole set of addresses. Actually, *merging triple* (defined below) will handle sets of addresses on which $\trianglelefteq$ is a tree-order.

Let us recall the example of Fig. 6. The algorithm starts with two different tokens $\mathtt{l}_1 = \{x\}$ and $\mathtt{l}_2 = \{y\}$ that are gathered in the address $\mathtt{A}$ which is the lowest element of the labeling. Each token corresponds to the lowest element of one of the simple nets. Step 3 introduces three new tokens $\mathtt{l}_1^1 = \{x, c_1\}$, $\mathtt{l}_2^1 = \{x, c_2\}$ and $\mathtt{l}_2^2 = \{x, c_3\}$ where c_1, c_2, c_3 correspond to the coderelictions in boldface on 6(c). These tokens are gathered in the address



(a) counter t such that $\mathfrak{L}(t) = \mathtt{A}_w$, $\mathfrak{L}(p_i) = \mathtt{1}_i$

(b) initial state $\mathrm{Init}_{\mathtt{A}_w}(\alpha_i, \mathtt{1}_i)_{i\leq n}$

Figure 8: counter and initial state.

$\mathtt{B} = \{\mathtt{1}_1^1, \mathtt{1}_2^1, \mathtt{1}_2^2\}$ which is used by Steps 4, 5 and 7. We have $\mathtt{A} \trianglelefteq \mathtt{B}$. Finally, Step 8 introduces the tokens $\mathtt{1}_1^{1,1} = \{x, c_1, d_1\}$, $\mathtt{1}_2^{1,1} = \{x, c_2, w\}$, $\mathtt{1}_2^{2,1} = \{x, c_3, d_2\}$ and $\mathtt{1}_2^{2,2} = \{x, c_3, d_3\}$ where d_1, w, d_2, d_3 are the coderelictions and coweakening in boldface on 6(h). The corresponding address $\mathtt{C} = \{\mathtt{1}_1^{1,1}, \mathtt{1}_2^{1,1}, \mathtt{1}_2^{2,1}, \mathtt{1}_2^{2,2}\}$ is used in Step 9. We have $\mathtt{B} \trianglelefteq \mathtt{C}$.

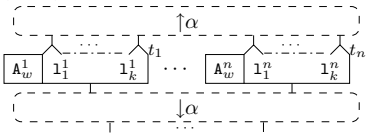
We are working with two orders on addresses: $\sqsubseteq$ and $\trianglelefteq$. The first encodes the structure of boxes, the second is built step by step by merging reduction. The reduction is defined so that the two orders coincide, even if they are defined differently. Indeed, $\trianglelefteq$ looks inside addresses, depending on the set-theoretical inclusion of the tokens (Def. 9), while $\sqsubseteq$ looks at the addresses as simple names labeling the cells of the net (Def. 7).

Definition 10. A **counter** is a cell t with one principal port and $n \geq 1$ auxiliary ports. Every port of t is labeled by the same MELL formula. We consider counters as commutative cells, i.e. their auxiliary ports are unordered. Moreover, t is given with a labeling function λ_t which maps every auxiliary port to a token and t itself to an address (Fig. 8(a)). We also require that for every ports p, q of t, $\lambda_t(p) and \lambda_t(q)$ are incomparable tokens and also are incomparable with every element of $\lambda_t(t)$.

In order to describe the partially labeled nets that appear during the reduction, we introduce triples.

Definition 11. A **triple** $(\alpha, \mathfrak{L}, \mathcal{A})$ is made of

- a simple net α with counters which can be decomposed into two counter-free simple nets $\downarrow\alpha$ and $\uparrow\alpha$ joined by counters $t_1, \ldots, t_n$ as follows:



- a set $\mathcal{A}$ of addresses and a function $\mathfrak{L}$ from $\mathrm{cell}(\downarrow\alpha) \cup \{t_1, \ldots, t_n\}$, $\bot_\alpha$ included, to $\mathcal{A}$, where for every counter t,
  $\mathfrak{L}(t) = \lambda_t(t) \cup \{\lambda_t(p) \,|\, p \text{ auxiliary port of } t\}$.

We are interested in reductions beginning on an initial state made of counters linking given simple nets.

Definition 12. Let $(\alpha_i)_{i\leq n}$ be a collection of simple nets with paired interfaces. Let $(\mathtt{1}_i)_{i\leq n}$ be tokens and $\mathtt{A}_w$ be an address such that the tokens in $\mathtt{A} = \mathtt{A}_w \cup \{\mathtt{1}_i\}_{i\leq n}$ are pairwise incomparable. The **initial state** associated with $(\alpha_i)_{i\leq n}$, $(\mathtt{1}_i)_{i\leq n}$, and $\mathtt{A}_w$ is the triple $(\alpha, \mathfrak{L}_\mathtt{A}, \mathcal{A})$ where
$\alpha = \mathrm{Init}_{\mathtt{A}_w}(\alpha_i, \mathtt{1}_i)_{i\leq n}$ is the simple net with counters pictured on Fig. 8(b) with $\mathfrak{L}_\mathtt{A}(t) = \mathtt{A}$ for every counter t and $\mathcal{A} = \{\mathtt{A}\}$. In the sequel, when $\mathtt{A}_w$ is empty, we will often omit the subscript and write $\mathrm{Init}(\alpha_i, \mathtt{1}_i)_{i\leq n}$.

Now we have all the ingredients to introduce a reduction $\xrightarrow{\mathtt{mrg}}$ on triples as the context closure of the binary relation $\mathtt{mrg}$ described in Fig. 9. In the interaction net paradigm [11], a redex is made of two cells wired by their principal ports. On the contrary, a merging redex is made of a counter whose *auxiliary* ports are linked to the principal ports of cells of simple nets. For this reason we represent the auxiliary ports of a counter as tips of triangles. It is important to notice that though the counters merge cells locally, the labeling process is global, whence the set of addresses appearing in the triples.

Definition 13. The **merging reduction** $\xrightarrow{\mathtt{mrg}}$ is the context closure of the **elementary reduction steps**, **ers** for short, in Fig. 9. This means $(\alpha_1, \mathfrak{L}_1, \mathcal{A}_1) \xrightarrow{\mathtt{mrg}} (\alpha_2, \mathfrak{L}_2, \mathcal{A}_2)$ if and only if there are two triples $(\alpha_1', \mathfrak{L}_1', \mathcal{A}_1), (\alpha_2', \mathfrak{L}_2', \mathcal{A}_2)$ s.t.:

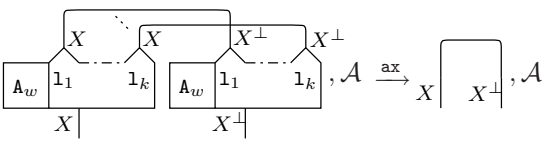
- $(\alpha_1', \mathfrak{L}_1', \mathcal{A}_1) \xrightarrow{\mathtt{x}} (\alpha_2', \mathfrak{L}_2', \mathcal{A}_2)$ for $\mathtt{x}$ an ers among $\mathtt{ax}$, $\mathtt{zero}$, $\mathtt{one}$, $\mathtt{two}$, $(\mathtt{co})\mathtt{w}$, $\mathtt{contr}$, $!\mathtt{p}$, $?\mathtt{p}$;
- $\alpha_1' \subseteq \alpha_1$ and $\alpha_2 = \alpha_1[\alpha_2'/\alpha_1']$;
- $\mathfrak{L}_1' = \mathfrak{L}_1|_{\mathrm{cell}(\alpha_1')}$, $\mathfrak{L}_2' = \mathfrak{L}_2|_{\mathrm{cell}(\alpha_2')}$ and for all $c \in \alpha_1 \setminus \alpha_1'$, $\mathfrak{L}_2(c) = \mathfrak{L}_1(c)$.

We denote by $\xrightarrow{\mathtt{mrg}*}$ the reflexive and transitive closure of $\xrightarrow{\mathtt{mrg}}$. We say that a reduction sequence $R : (\alpha, \mathfrak{L}, \mathcal{A}) \xrightarrow{\mathtt{mrg}*} (\alpha', \mathfrak{L}', \mathcal{A}')$ is **successful** if α' is counter-free.

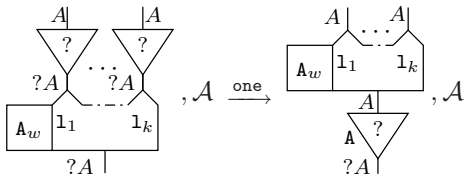
Notice Fig. 9 has three non-deterministic ers $(\mathtt{contr}, !\mathtt{p}, ?\mathtt{p})$, where one can choose different surjections $(\mathtt{contr}, ?\mathtt{p})$ or addresses $(!\mathtt{p}, ?\mathtt{p})$. Actually the non-determinism of $!\mathtt{p}$ is irrelevant, since different choices of $\mathtt{A}'$ yield the same labeling, modulo a renaming.

There are only three cases where a reduction falls into a **deadlock**, i.e. stops on a triple no further reducible but with counters: when two counters are linked by axioms and have different labels; when a counter is linked to one axiom and another cell; when there is no possible address to go through a contraction link for a $?\mathtt{p}$-ers.

Since we want the reduction to produce a labeling, we have to restrict the set of triples that we consider. So we introduce merging triples such that the result of a successful reduction (i.e. a counter-free merging triple) is a labeling. Then we prove that the reduction preserves the properties of the merging triples. Since the initial states are merging triples and the counter-free merging triples are labelings, we get the wanted result.

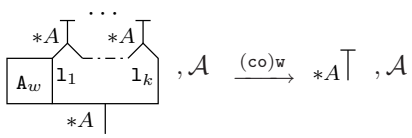


in case $*$ is one between $1, \bot$, and $\mathtt{A}_w \cup \{\mathtt{l}_i\}_{i\leq k} = \mathtt{A} \in \mathcal{A}$.

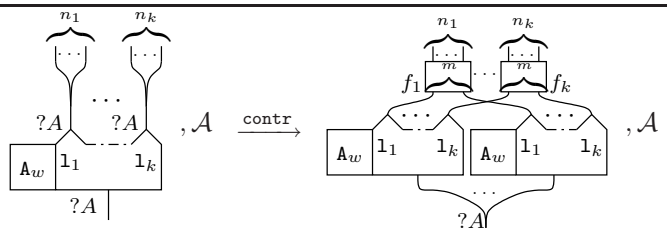


in case $*$ is one between $\parr, \otimes$, and $\mathtt{A}_w \cup \{\mathtt{l}_i\}_{i\leq k} = \mathtt{A} \in \mathcal{A}$.

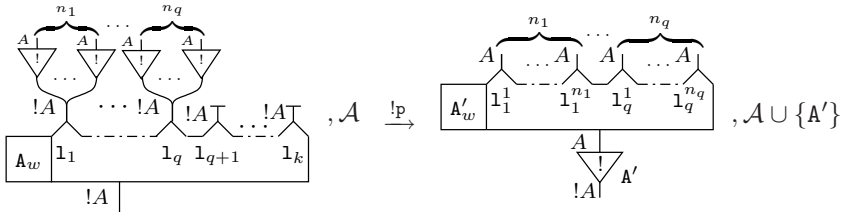
in case $\mathtt{A}_w \cup \{\mathtt{l}_i\}_{i\leq k} = \mathtt{A} \in \mathcal{A}$.



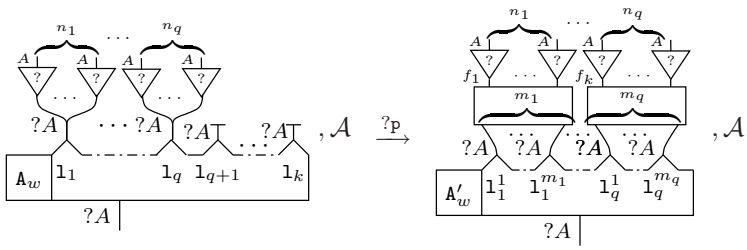
in case $*$ is one between ?, !.



where we choose a number m, such that $2 \leq m \leq \min_{i\leq k} n_i$, and a surjection f_i from n_i to m, for every $i \leq k$.



we order the ports of the counter in the redex so that the first $q \geq 0$ (resp. last $k-q$) ports are connected to coderelictions (resp. coweakenings), this order is just a convention to have a smooth indexing; we then choose a family $\{y\} \cup \{x_i^j\}_{i\leq q, j\leq n_i}$ of pairwise distinct elements of X which are fresh in $\mathcal{A}$, and we set $\mathtt{A}'_w = \mathtt{A}_w \cup \{\mathtt{l}_i \cup \{y\}\}_{q<i\leq k}$ and $\mathtt{A}' = \mathtt{A}'_w \cup \{\mathtt{l}_i^j\}_{i\leq q, j\leq n_i}$, where $\mathtt{l}_i^j = \mathtt{l}_i \cup \{x_i^j\}$. We say that this ers **creates** the address $\mathtt{A}'$.



like in the ers !p, we order the ports of the counter in the redex so that the first $q \geq 0$ (resp. last $k-q$) ports are connected to derelictions (resp. weakenings); then we choose a number $m_i \leq n_i$ and a surjection f_i from n_i to m_i for every $i \leq q$; we choose $\mathtt{A}' \in \mathcal{A}$ successor of the labeling of the counter in the redex (if $\mathcal{A}$ has no successor then ?p cannot be applied), i.e. such that there exists a family $\{y\} \cup \{x_i^j\}_{i\leq q, j\leq n_i}$ of pairwise distinct elements of X s.t. $\mathtt{A}'_w = \mathtt{A}_w \cup \{\mathtt{l}_i \cup \{y\}\}_{q<i\leq k}$ and $\mathtt{A}' = \mathtt{A}'_w \cup \{\mathtt{l}_i^j\}_{i\leq q, j\leq m_i}$, where $\mathtt{l}_i^j = \mathtt{l}_i \cup \{x_i^j\}$. We say this ers **enters** in the address $\mathtt{A}'$.

Figure 9: the **elementary reduction steps** (**ers** for short) of merging reduction; the net at left of an ers is the **redex**, the one at right is the **contractum** of the ers. In the ers contr, ?p, we present a bunch of contractions and wirings as a surjective function f from the auxiliary ports to the principal ones.

Definition 14. The triple $(\alpha, \mathfrak{L}, \mathcal{A})$ is called a **merging triple** if it satisfies

(i) for every counter $t \in \alpha$, the principal port of t is wired to a cell $c \in \downarrow\alpha$, $\bot_\alpha$ included, and $\mathfrak{L}(c) \trianglelefteq \mathfrak{L}(t)$; moreover, if $\mathfrak{L}(c) \neq \mathfrak{L}(t)$ then every auxiliary port of t is wired to a dereliction $c \in \uparrow\alpha$, or a weakening;
(ii) $\forall \mathtt{A}, \mathtt{B} \in \mathcal{A}$, $(\mathtt{A} \sqsubseteq_{\downarrow\alpha, \mathfrak{L}} \mathtt{B} \iff \mathtt{A} \trianglelefteq \mathtt{B})$,
(iii) $(\downarrow\alpha, \mathfrak{L})$ is a labeling and $\mathfrak{L}(\downarrow\alpha) = \mathcal{A}$.

Notice that the initial state (Def. 12) is a merging triple.

As we wrote above $\trianglelefteq$ is not in general an order; however, if $(\alpha, \mathfrak{L}, \mathcal{A})$ is a merging triple, then Cond. (ii), (iii) guarantees that $\trianglelefteq$ is a tree-order on $\mathcal{A}$.

Proposition 2. *If $(\alpha, \mathfrak{L}, \mathcal{A}) \xrightarrow{\mathtt{mrg}} (\alpha', \mathfrak{L}', \mathcal{A}')$ and $(\alpha, \mathfrak{L}, \mathcal{A})$ is a merging triple, then $(\alpha', \mathfrak{L}', \mathcal{A}')$ is also a merging triple.*

As a consequence, if a reduction $(\alpha, \mathfrak{L}, \mathcal{A}) \xrightarrow{\mathtt{mrg}*} (\alpha', \mathfrak{L}', \mathcal{A}')$ is successful, $(\alpha', \mathfrak{L}') = (\downarrow\alpha', \mathfrak{L}')$ is a labeling and so represents an ll-net π (Prop. 1). In this case, we say that the reduction **leads to** π and we write

$$(\alpha, \mathfrak{L}, \mathcal{A}) \xrightarrow{\mathtt{mrg}*} \pi.$$

Proposition 3. *The number of ers of any $\xrightarrow{\mathtt{mrg}}$-reduction starting from a merging triple $(\alpha, \mathfrak{L}, \mathcal{A})$ is polynomially bounded by the number of ports in α.*

The ers of Fig. 9 are local and can be implemented on a Turing Machine in constant time. Thus Prop. 3 yields:

Corollary 4. *The runtime of any $\xrightarrow{\mathtt{mrg}}$-reduction starting from a merging triple $(\alpha, \mathfrak{L}, \mathcal{A})$ is polynomial in the number of ports in α.*

D. Completeness and soundness

We prove the completeness and soundness of merging reduction. The soundness ensures that the simple nets merged into an ll-net, are in the Taylor expansion of this ll-net. The completeness theorem proves the converse: simple nets $\{\alpha_i\}_{i\leq n}$ coming from the Taylor expansion of a same ll-net π can be merged. In fact, since $\{\alpha_i\}_{i\leq n}$ is a *finite* subset of $\mathcal{T}(\pi)$, then some boxes of π can remain undefined from the merging of $\{\alpha_i\}_{i\leq n}$. Formally this means that the merging yields an ll-net less informative than π (Def. 6)

Theorem 1 (Completeness)**.** *Let π be an ll-net, and let $\{\alpha_i\}_{i\leq n}$ be simple nets in $\mathcal{T}(\pi)$. For any family $\{\mathtt{l}_i\}_{i\leq n} = \mathtt{A}$ of tokens pairwise incomparable, there exists an ll-net $\pi_0 \ll \pi$ and a successful reduction that leads to π_0:*

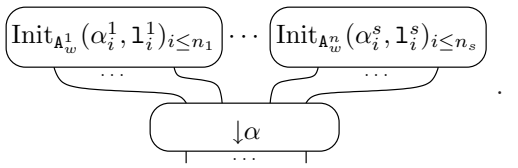
$$(\mathrm{Init}(\alpha_i, \mathtt{l}_i)_{i\leq n}, \mathfrak{L}_\mathtt{A}, \{\mathtt{A}\}) \xrightarrow{\mathtt{mrg}*} \pi_0,$$

where $\mathfrak{L}_\mathtt{A}$ is, as usual, the constant function taking value $\mathtt{A}$.

Proof: The proof is by induction on the exponential depth of π; we split the induction step in two cases: if π is a box, we choose tokens extending the initial ones and gather them in an address $\mathtt{B}$. We apply the induction hypothesis using the address $\mathtt{B}$ and conclude by context closure; in the general case, we make counter go through the linear part, and stop at the entrance of boxes. We use the one box case and conclude by context closure. ■

To prove the soundness theorem, we need a splitting lemma which decomposes $\uparrow\alpha$ in initial states (Def. 12).

Lemma 5 (Splitting)**.** *Let R be a successful reduction sequence from a merging triple $(\alpha, \mathfrak{L}, \mathcal{A})$ and s.t. no ers of R enters an address labeling a counter of α; then α can be split: there are suitable sequences $(\alpha_i^r)_{r\leq s, i\leq n_r}$ of simple nets, $(\mathtt{l}_i^r)_{r\leq s, i\leq n_r}$ of tokens pairwise incomparable and $(\mathtt{A}_w^r)_{r\leq s}$ of addresses s.t.*

$\alpha \;=\;$ 

Proof: First we prove that since R is successful, two counters of the same connected component of $\uparrow\alpha$ have the same label. With each label $\mathtt{A}$, we associate the subnet β made of cells connected to a counter labeled by $\mathtt{A}$. Then, two connected auxiliary ports of counters have the same label. This allows to decompose β into an initial state. ■

Theorem 2 (Soundness)**.** *Let π be an ll-net, let $(\alpha_i)_{i\leq n}$ be a family of simple nets with the same interface and let $(\mathtt{l}_i)_{i\leq n} = \mathtt{A}$ be a family of pairwise incomparable tokens. If there is a successful merging reduction leading to π:*

$$(\mathrm{Init}(\alpha_i, \mathtt{l}_i)_{i\leq n}, \mathfrak{L}_\mathtt{A}, \{\mathtt{A}\}) \xrightarrow{\mathtt{mrg}*} \pi \qquad (3)$$

then for every $i \leq n$, $\alpha_i \in \mathcal{T}(\pi)$.

Proof: The proof is by induction on the exponential depth of π. The main idea is to commute the ers in the reduction leading to π and to gather the ers that enter a box. Thanks to the splitting Lemma 5, we get an initial state for each box. Thus, we can apply the induction hypothesis. ■

III. Perspectives

The merging reduction could have been presented differently, gathering all the non-deterministic choices in a single initial step and then performing the "deterministic" part of the reduction. This amounts to choose a labeling $\mathfrak{L}$ on the upper part $\uparrow\alpha$ of an initial state and to transform the merging reduction in a rewriting that checks deterministically whether $\mathfrak{L}$ is correct. However we have preferred the most local presentation of the reconstruction of the boxes.

A polynet $\{\alpha_i\}_{i\leq n}$ included in the Taylor expansion of an ll-net π is *uniform* in the sense that every α_i is built from the same ll-net π, by taking different numbers of copies of the boxes in π. Our algorithm yields a combinatorial criterion characterizing the polynets of DiLL which are uniform. Ehrhard and Laurent have used the sum constructor

in DiLL to model concurrent computing [12]. Merging reduction opens a logical approach to a typical problem of synchronization: when are processes uniform and can be join up?

The informative order of Def. 6 suggests a natural question: is the Taylor expansion injective? This amounts to question whether the converse of the implication (1) of Sect. I holds. Indeed if two ll-nets π, π' are different, then either $\pi' \not\ll \pi$ or $\pi \not\ll \pi'$, hence from the converse of the implication (1) we would have $\mathcal{T}(\pi') \neq \mathcal{T}(\pi)$.

Till now, the first author and Mazza [13] have shown the equivalence between the injectivity of the Taylor expansion and the injectivity of the LL relational semantics for cut-free LL nets,[5] which is an open problem first addressed in [14]. Our results would allow to deduce the injectivity from the following confluence property of the merging reduction:

$(*)$ for every cut-free ll-net π', there are simple nets $(\alpha_i)_{i\leq n} \in \mathcal{T}(\pi')$ s.t. every successful reduction leads to π': $\mathrm{Init}(\alpha_i)_{i\leq n} \xrightarrow{\texttt{mrg*}} \pi'$.

Let us show how $(*)$ yields $\mathcal{T}(\pi') \subseteq \mathcal{T}(\pi) \Rightarrow \pi' \ll \pi$, the converse of (1). Let π' and π be ll-nets s.t. $\mathcal{T}(\pi') \subseteq \mathcal{T}(\pi)$. Assume $(*)$: there are $(\alpha_i)_{i\leq n}$ in $\mathcal{T}(\pi')$ s.t. every succesful reduction leads to π'. Since $(\alpha_i)_{i\leq n} \subseteq \mathcal{T}(\pi)$, by completeness (Th. 1) there is $\pi_0 \ll \pi$ such that $\mathrm{Init}(\alpha_i)_{i\leq n} \xrightarrow{\texttt{mrg*}} \pi_0$. By $(*)$, $\pi_0 = \pi'$, so $\pi' \ll \pi$.

Acknowledgement.

We are grateful to O. Laurent for having suggested counters travelling through nets and to T. Ehrhard and P.-L. Curien for useful discussions and hints.

References

[1] J.-Y. Girard, "Linear logic," *Theor. Comput. Sci.*, vol. 50, pp. 1–102, 1987.

[2] ——, "Normal functors, power series and λ-calculus," *Annals of Pure and Applied Logic*, vol. 37, no. 2, pp. 129–177, 1988.

[3] ——, "Coherent banach spaces: a continuous denotational semantics," *Theor. Comput. Sci.*, vol. 3, pp. 227–275, 1996.

[4] R. Hasegawa, "Two applications of analytic functors," *Theor. Comput. Sci.*, vol. 272, no. 1-2, pp. 113–175, 2002.

[5] T. Ehrhard, "Finiteness spaces," *Math. Struct. Comput. Sci.*, vol. 15, no. 4, pp. 615–646, 2005.

[6] T. Ehrhard and L. Regnier, "Differential interaction nets," *Theor. Comput. Sci.*, vol. 364, no. 2, pp. 166–195, 2006.

[7] ——, "The differential lambda-calculus," *Theor. Comput. Sci.*, vol. 309, no. 1-3, pp. 1–41, 2003.

[8] ——, "Uniformity and the Taylor expansion of ordinary lambda-terms," *Theor. Comput. Sci.*, 2008.

[5] This equivalence has been remarked in a private discussion with Daniel De Carvalho, too.

[9] L. Regnier, "Lambda-calcul et réseaux," Thèse de Doctorat, Université Paris VII, 1992.

[10] L. Vaux, "λ-calcul différentiel et logique classique : interactions calculatoires," Thèse de Doctorat, Université Aix-Marseille II, 2007.

[11] Y. Lafont, "Interaction nets," in *POPL '90: Proceedings*. New York, NY, USA: ACM, 1990, pp. 95–108.

[12] T. Ehrhard and O. Laurent, "Interpreting a finitary pi-calculus in differential interaction nets," in *Concurrency Theory (CONCUR '07)*, ser. Lecture Notes in Comput. Sci., vol. 4703. Springer, Sep. 2007, pp. 333–348.

[13] D. Mazza and M. Pagani, "The separation theorem for differential interaction nets," in *LPAR*, ser. Lecture Notes in Comput. Sci., vol. 4790. Springer, 2007, pp. 393–407.

[14] L. Tortora de Falco, "Obsessional experiments for linear logic proof-nets," *Math. Struct. Comput. Sci.*, vol. 13, no. 6, pp. 799–855, 2003.